

# Tipo Abstrato de Dados(TAD), Ponteiros e Exercícios

Sergio Canuto  
[sergio.canuto@ifg.edu.br](mailto:sergio.canuto@ifg.edu.br)

## Na aula passada...

1) Implemente um código em C capaz de alocar com malloc um vetor de 4 números do tipo inteiro. Depois, o programa deve solicitar ao usuário a entrada dos 4 números no espaço alocado. Por fim, o programa deve mostrar os 4 números e liberar a memória alocada. (solução do aluno:)

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *ptr = NULL;
    ptr = (int*) malloc(4*sizeof(int));
    if(ptr==0)
        exit(1);
    printf("Digite 4 numeros inteiros:\n");
    for(int i=0; i<4;i++)
    {
        scanf("%d", ptr+i);
    }
    printf("\nVoce digitou:\n");
    for(int i=0; i<4;i++)
    {
        printf("%d ", *(ptr+i));
    }
    free(ptr);
    return 0;
}
```

## Na aula passada...

2) Implemente um código em C que inicialmente recebe um inteiro (com scanf) que será usado como tamanho de uma string. Depois, o código deve alocar dinamicamente uma string com o tamanho definido, e em seguida, o conteúdo dessa string deve ser preenchido pelo usuário (também com scanf). O programa deve imprimir o conteúdo da string sem seus caracteres numéricos. (solução do aluno:)

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int main ()
{
    int tamanho;
    char *ptr;
    printf("Informe o tamanho da string desejada:\n");
    scanf("%d", &tamanho);
    ptr = (char*)malloc(tamanho*sizeof(char));
    if(ptr==0)
        exit(1);
    printf("\n\nEscreva uma string de ate %d caracteres:\n", tamanho);
    scanf("%s", ptr);
    printf("\n\nString filtrada:\n");
    for (int i = 0; i<tamanho;i++)
    {
        if (isdigit(*(ptr+i)) == 0) //verifica se o caracter eh ou nao um numero
        {
            printf("%c", *(ptr+i)); //imprime caracter por caracter
        }
    }
    free(ptr);
    return 0;
}
```

## Na aula passada...

3) Implemente um código em C que declare uma struct para o cadastro de trabalhadores de uma empresa. No código:

- a) Deverão ser armazenados, para cada trabalhador: cpf, nome e ano de nascimento.
- b) Primeiramente, o usuário deverá inserir o número de trabalhadores que serão armazenados
- c) O código deverá alocar dinamicamente a quantidade necessária de memória para armazenar os registros dos trabalhadores.
- d) O código deverá solicitar a inserção das informações dos trabalhadores
- e) Por fim, deverá ser impressa na tela os dados armazenados e deverá ser feita a liberação da memória.\* /

## Na aula passada (solução do aluno 1)...

```
#include <stdio.h>
#include <stdlib.h>
```

```
/typedef struct{
    char cpf[15];
    char nome[30];
    int anoNascimento;
} cadastro;
```

```
int main()
{
    cadastro *ptr;
    int totalTrabalhadores;
```

```
    printf("\nInforme o numero de trabalhadores:\n");
    scanf("%d", &totalTrabalhadores);
```

```
    ptr = (cadastro*)malloc(totalTrabalhadores*sizeof(cadastro));
```

```
    for(int i = 0; i<totalTrabalhadores; i++)
```

```
    {
        printf("\n\tCadastro Funcionario 00%d\n", i+1);
        printf("CPF (###.###.###-##): ");
        scanf(" %s", ptr[i].cpf);
        printf("Nome: ");
        scanf(" %s", ptr[i].nome);
        printf("Ano de nascimento (####): ");
        scanf("%d", &ptr[i].anoNascimento);
```

```
    }
```

```
    printf("\n\t\tFUNCIONARIOS CADASTRADOS\n");
```

```
    for(int i = 0; i<totalTrabalhadores; i++)
```

```
    {
        printf("\n\tCadastro Funcionario 00%d\n", i+1);
        printf("CPF: %s\n", ptr[i].cpf);
        printf("Nome: %s\n", ptr[i].nome);
        printf("Ano de nasciment: %d\n", ptr[i].anoNascimento);
    }
```

```
    free(ptr);
    return 0;
```

```
}
```

## Na aula passada (solução do aluno 2)...

```
#include <stdlib.h>
#include <stdio.h>

typedef struct {
    int dia;
    int mes;
    int ano;
} data;

typedef struct {
    long int cpf;
    char *nome;
    data nascimento;
} trabalhador;

void quest3() {
    int pessoas;
    printf("Digite a quantidade de pessoas desejada:\n");
    scanf("%d", &pessoas);
    trabalhador *trabalhadores = (trabalhador *)malloc(pessoas * sizeof(trabalhador));
    for (int i = 0; i < pessoas; i++) {
        printf("Trabalhador %d:\n", i + 1);
        printf("CPF: ");
        scanf("%ld", &trabalhadores[i].cpf);
        printf("Nome: ");
        char nome[100]; // Assumindo que o nome não terá mais de 100 caracteres
        scanf(" %99[^\n]", nome); // Lê até 99 caracteres (evita o problema do \n no buffer)
        trabalhadores[i].nome = strdup(nome);
        printf("Data de Nascimento (dia mes ano): ");
        scanf("%d %d %d", &trabalhadores[i].nascimento.dia,
&trabalhadores[i].nascimento.mes, &trabalhadores[i].nascimento.ano);
        printf("\n");
    }
}
```

```
printf("Trabalhadores cadastrados:\n");
for (int i = 0; i < pessoas; i++) {
    printf("Trabalhador %d:\n", i + 1);
    printf("CPF: %ld\n", trabalhadores[i].cpf);
    printf("Nome: %s\n", trabalhadores[i].nome);
    printf("Data de Nascimento: %02d/%02d/%04d\n",
        trabalhadores[i].nascimento.dia,
        trabalhadores[i].nascimento.mes,
        trabalhadores[i].nascimento.ano);
    printf("\n");
}

for (int i = 0; i < pessoas; i++) {
    free(trabalhadores[i].nome);
}
free(trabalhadores);
}
```

# Conceituações - Tipo Abstrato de dados

- Um **tipo de dado** define o conjunto de **valores** (domínio) e **operações** que uma variável pode assumir.
  - Ex.: Int, Char, float, etc...
- Uma **estrutura de dados** consiste em um conjunto de tipos de dados em que existe algum tipo de relacionamento lógico estrutural.
  - Ex.: Na linguagem C temos os **array**, **struct**, **union** e **enum**, todas criadas a partir dos tipos de dados básicos.
- Um **tipo abstrato de dados**, ou **TAD**, é um conjunto de dados estruturados e as operações que podem ser executadas sobre esses dados.
- Tanto a **representação** quanto **as operações** do **TAD** são especificadas pelo programador.
  - O usuário utiliza o **TAD** como uma caixa-preta por meio de sua **interface**.

# Tipo Abstrato de dados - motivação

## Exemplo: estrutura do tipo FILE

```
01 typedef struct{
02     int      level;          // nível do buffer
03     unsigned flags;          // flag de status do arquivo
04     char      fd;             // descritor do arquivo
05     unsigned char hold;      // retorna caractere se sem buffer
06     int       bsize;          // tamanho do Buffer
07     unsigned char *buffer;    // buffer de transferência de dados
08     unsigned char *curp;      // ponteiro atualmente ativo
09     unsigned   istemp;        // indicador de arquivo temporário
10     short      token;         // usado para validação
11 }FILE;
12
```

Alguns acreditam que ninguém, em sã consciência, deve fazer uso direto dos campos dessa estrutura!



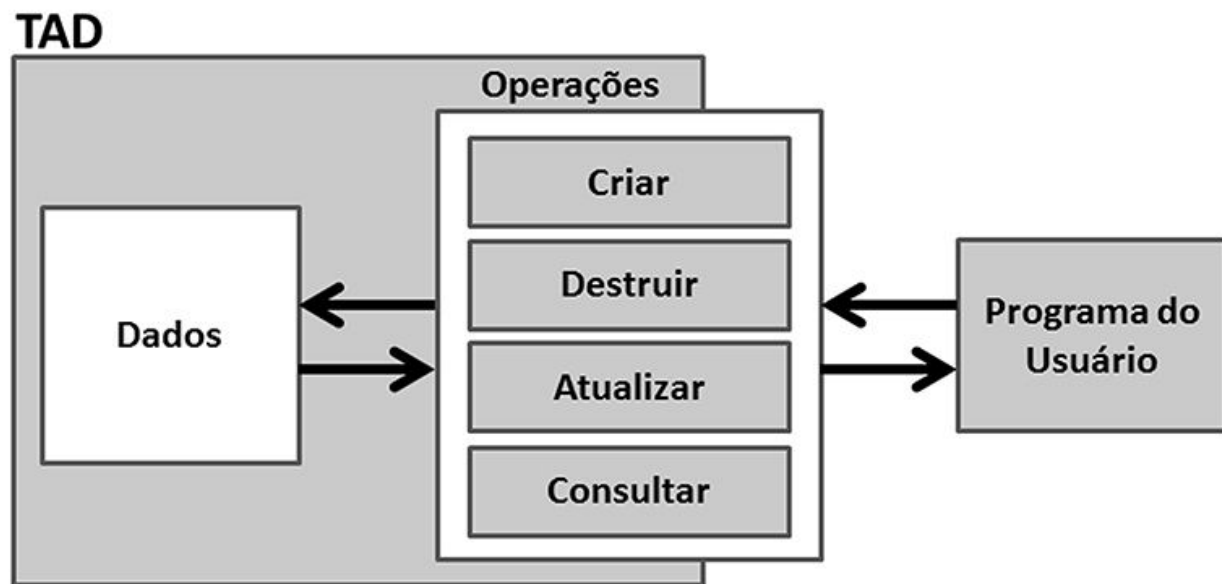
# Tipo Abstrato de dados

- Então, a única maneira de trabalhar com arquivos em linguagem C é declarando um ponteiro de arquivo da seguinte maneira:
  - `FILE* f;`
- Desse modo, o usuário possui apenas um ponteiro para onde os dados estão armazenados, mas não pode acessá-los diretamente.
- A única maneira de acessar o conteúdo do ponteiro FILE é por meio das operações definidas em sua interface.
- Assim, os dados do ponteiro f somente podem ser acessados pelas funções de manipulação do tipo FILE:
  - `fopen()`
  - `fclose()`
  - `fputc()`
  - `fgetc()`
  - `etc...`

# Tipo Abstrato de dados - Operações Básicas

- Tipos abstratos de dados incluem as operações para a manipulação de seus dados. Essas operações variam de acordo com o **TAD** criado, porém as seguintes operações básicas são possíveis:
  - Criação do **TAD**.
  - Inserção de um novo elemento no **TAD**.
  - Remoção de um elemento do **TAD**.
  - Acesso a um elemento do **TAD**.
  - Destruição do **TAD**.

# Tipo Abstrato de dados



## Tipo Abstrato de dados

O uso de um TAD traz uma série de vantagens:

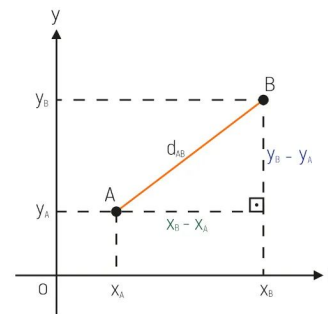
- Encapsulamento: ao ocultarmos a implementação, fornecemos um conjunto de operações possíveis para o TAD.
- Segurança: o usuário não tem acesso direto aos dados. Isso evita que ele manipule os dados de uma maneira imprópria.
- Flexibilidade: podemos alterar o TAD sem alterar as aplicações que o utilizam.
- Reutilização: a implementação do TAD é feita em um módulo diferente do programa do usuário.

# Questão - Tipo Abstrato de Dados

Seja o seguinte código que se propõe a calcular a distância entre dois pontos:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    struct ponto{
        float x;
        float y;
    };
    typedef struct ponto Ponto;
    Ponto p1, p2;
    p1.x=2; p1.y=3; p2.x=5; p2.y=8;
    float dx = p1.x - p2.x;
    float dy = p1.y - p2.y;
    printf ("distancia: %f\n",sqrt(dx * dx + dy * dy));
    return 0;
}
```

Distância euclidiana:



$$d_{AB} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

Considerando o código acima, analise as afirmações a seguir em relação às características do tipo abstrato de dados para o cálculo de distâncias:

I - Possui a característica de encapsulamento para o cálculo de distâncias.

II - Não há acesso direto aos dados, portanto, possui a característica de segurança.

III - Possui a característica de flexibilidade, pois é possível alterar o tipo abstrato de dados sem alterar as aplicações que o utilizam.

IV - O código do cálculo de distância não possui as características que regem o tipo abstrato de dados.

# Questão - Tipo Abstrato de Dados

Seja o seguinte código que se propõe a calcular a distância entre dois pontos:

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    struct ponto{
        float x;
        float y;
    };
    typedef struct ponto Ponto;
    Ponto p1, p2;
    p1.x=2; p1.y=3; p2.x=5; p2.y=8;
    float dx = p1.x - p2.x;
    float dy = p1.y - p2.y;
    printf ("distancia: %f\n",sqrt(dx * dx + dy * dy));
    return 0;
}
```

Considerando o código acima, analise as afirmações a seguir em relação às características do tipo abstrato de dados para o cálculo de distâncias:

I - Possui a característica de encapsulamento para o cálculo de distâncias.

II - Não há acesso direto aos dados, portanto, possui a característica de segurança.

III - Possui a característica de flexibilidade, pois é possível alterar o tipo abstrato de dados sem alterar as aplicações que o utilizam.

**IV - O código do cálculo de distância não possui as características que regem o tipo abstrato de dados.**

# Questão - Tipo Abstrato de Dados

12) Em relação aos tipos abstratos de dados TAD, é correto afirmar:

- a) O TAD não encapsula a estrutura de dados para permitir que os usuários possam ter acesso a todas as operações sobre esses dados.
- b) Alterações na implementação de um TAD implicam em alterações em seu uso.
- c) os tipos abstratos de dados podem ser formados pela união de tipos de dados primitivos, mas não por outros tipos abstratos de dados.
- d) TAD é um tipo de dados que esconde a sua implementação de quem o manipula; de maneira geral as operações sobre estes dados são executadas sem que se saiba como isso é feito.

# Questão - Tipo Abstrato de Dados

12) Em relação aos tipos abstratos de dados TAD, é correto afirmar:

- a) O TAD não encapsula a estrutura de dados para permitir que os usuários possam ter acesso a todas as operações sobre esses dados.
- b) Alterações na implementação de um TAD implicam em alterações em seu uso.
- c) os tipos abstratos de dados podem ser formados pela união de tipos de dados primitivos, mas não por outros tipos abstratos de dados.
- d) **TAD é um tipo de dados que esconde a sua implementação de quem o manipula; de maneira geral as operações sobre estes dados são executadas sem que se saiba como isso é feito.**



# Exemplo de TAD - Ponto

## Arquivo Ponto.h

```
01  typedef struct ponto Ponto;  
02  //Cria um novo ponto  
03  Ponto* Ponto_cria(float x, float y);  
04  //Libera um ponto  
05  void Ponto_libera(Ponto* p);  
06  //Acessa os valores "x" e "y" de um ponto  
07  int Ponto_acessa(Ponto* p, float* x, float* y);  
08  //Atribui os valores "x" e "y" a um ponto  
09  int Ponto_atribui(Ponto* p, float x, float y);  
10  //Calcula a distância entre dois pontos  
11  float Ponto_distancia(Ponto* p1, Ponto* p2);
```

## Arquivo Ponto.c

```
01  #include <stdlib.h>  
02  #include <math.h>  
03  #include "Ponto.h" //inclui os Protótipos  
04  struct ponto{//Definição do tipo de dados  
05      float x;  
06      float y;  
07  };
```

# Exemplo de TAD - Ponto

## Criando um ponto

```
01 Ponto* Ponto_cria(float x, float y){
02     Ponto* p = (Ponto*) malloc(sizeof(Ponto));
03     if(p != NULL){
04         p->x = x;
05         p->y = y;
06     }
07     return p;
08 }
```

## Destruindo um ponto

```
01 void Ponto_libera(Ponto* p){
02     free(p);
03 }
```

## Exemplo de TAD - Ponto

### Atribuindo um valor ao ponto

```
01  int Ponto_atribui(Ponto* p, float x, float y){
02      if(p == NULL)
03          return 0;
04      p->x = x;
05      p->y = y;
06      return 1;
07  }
```

### Calculando a distância entre dois pontos

```
01  float Ponto_distancia(Ponto* p1, Ponto* p2){
02      if(p1 == NULL || p2 == NULL)
03          return -1;
04      float dx = p1->x - p2->x;
05      float dy = p1->y - p2->y;
06      return sqrt(dx * dx + dy * dy);
07  }
```

# Exemplo de TAD - main\_ponto.c

## Exemplo: utilizando o TAD ponto

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include "Ponto.h"
04 int main() {
05     float d;
06     Ponto *p,*q;
07     //Ponto r; //ERRO
08     p = Ponto_cria(10,21);
09     q = Ponto_cria(7,25);
10     //q->x = 2; //ERRO
11     d = Ponto_distancia(p,q);
12     printf("Distancia entre pontos: %f\n",d);
13     Ponto_libera(q);
14     Ponto_libera(p);
15     system("pause");
16     return 0;
17 }
```

# Exemplo de TAD - Ponto

Compilando (na mão!)

//no console, entre no diretório onde estão os códigos:

**cd diretorio\_dos\_codigos**

//Compilando a biblioteca:

**gcc -c Ponto.c -o Ponto.o**

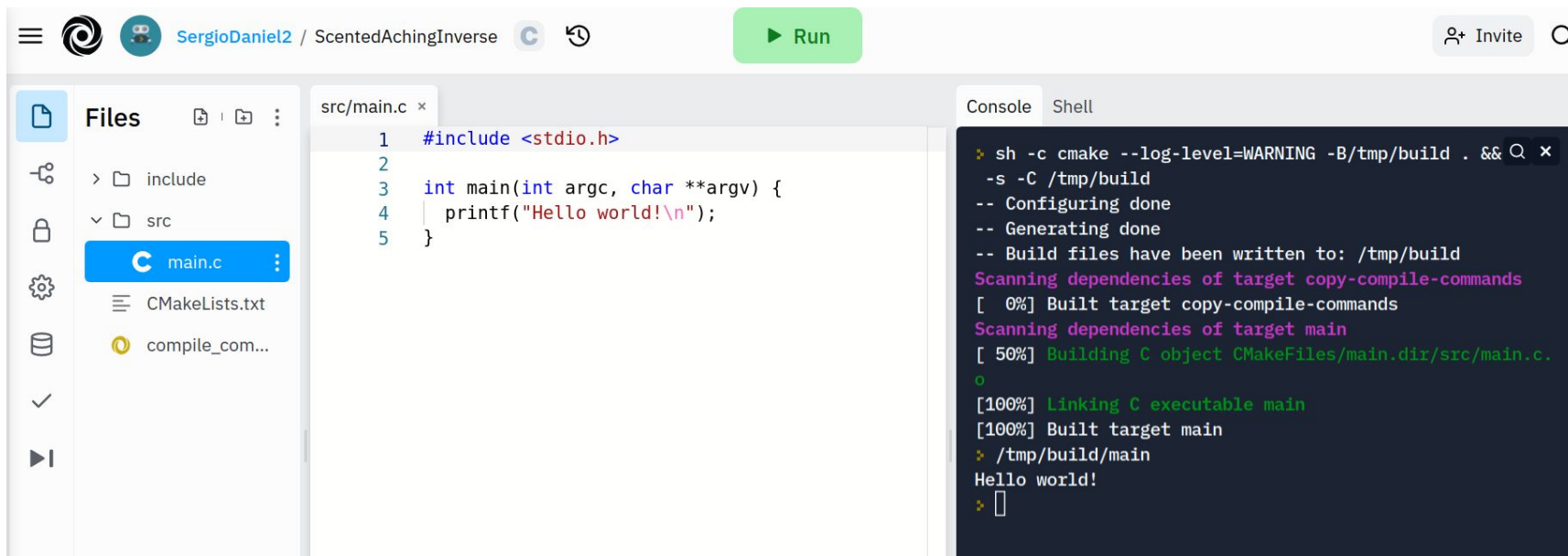
//Criando o executável a partir do main\_ponto.c e biblioteca compilada:

**gcc -o ponto.exe main\_ponto.c Ponto.o**

# Criando uma TAD

Desenvolva um TAD que represente um quadrado. Inclua as funções de inicialização necessárias e as operações que retornem a área e seu perímetro.

Ambiente: repl.it.com



The screenshot displays the Repl.it online IDE interface. At the top, the user profile 'SergioDaniel2' and the repository name 'ScentedAchingInverse' are visible. A green 'Run' button is present. The left sidebar shows a file explorer with a 'src' directory containing 'main.c'. The main editor window shows the following C code in 'src/main.c':

```
1 #include <stdio.h>
2
3 int main(int argc, char **argv) {
4     printf("Hello world!\n");
5 }
```

On the right, the 'Console' tab shows the output of the compilation and execution:

```
❯ sh -c cmake --log-level=WARNING -B/tmp/build . && Q x
-s -C /tmp/build
-- Configuring done
-- Generating done
-- Build files have been written to: /tmp/build
Scanning dependencies of target copy-compile-commands
[ 0%] Built target copy-compile-commands
Scanning dependencies of target main
[ 50%] Building C object CMakeFiles/main.dir/src/main.c.o
[100%] Linking C executable main
[100%] Built target main
❯ /tmp/build/main
Hello world!
❯
```

# Criando uma TAD

Desenvolva um TAD que represente um quadrado. Inclua as funções de inicialização necessárias e as operações que retornem a área e seu perímetro.

[http://scanuto.com/main\\_quadrado.c](http://scanuto.com/main_quadrado.c)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "Quadrado.h"
4
5 int main(){
6     float area, perimetro;
7     Quadrado* c1;
8     c1=quadrado_cria(2.0);
9     area = quadrado_area(c1);
10    printf("A area do quadrado eh: %f\n", area);
11    perimetro=quadrado_perimetro(c1);
12    printf("O perimetro do quadrado eh: %f\n", perimetro);
13
14    return 0;
15
16 }
```

# Criando uma TAD

Desenvolva um TAD que represente um quadrado. Inclua as funções de inicialização necessárias e as operações que retornem a área e seu perímetro.

[http://scanuto.com/main\\_quadrado.c](http://scanuto.com/main_quadrado.c)

O que deve ser definido em no TAD do Quadrado.h?

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "Quadrado.h"
4
5 int main(){
6     float area, perimetro;
7     Quadrado* c1;
8     c1=quadrado_cria(2.0);
9     area = quadrado_area(c1);
10    printf("A area do quadrado eh: %f\n", area);
11    perimetro=quadrado_perimetro(c1);
12    printf("O perimetro do quadrado eh: %f\n", perimetro);
13
14    return 0;
15
16 }
```



# Criando uma TAD

Desenvolva um TAD (em um novo arquivo Quadrado.c) que represente um quadrado. Inclua as funções de inicialização necessárias e as operações que retornem a área e seu perímetro.

<http://scanuto.com/Quadrado.h>

```
1 //arquivo Quadrado.h
2
3 typedef struct quadrado Quadrado;
4
5 //Cria um novo quadrado
6 Quadrado* quadrado_cria(float a);
7
8 //libera quadrado
9 void quadrado_libera(Quadrado* c);
10
11 //Acessa o valor "a" de quadrado
12 float quadrado_acessa(Quadrado* c);
13
14 // Atribui o valor a de um quadrado
15 void quadrado_atribui(Quadrado* c, float a);
16
17 // calcula area do quadrado
18 float quadrado_area(Quadrado* c);
19
20 //calcula o perimetro de um quadrado
21 float quadrado_perimetro(Quadrado* c);
```

# Criando uma TAD

Desenvolva um TAD que represente um polígono. Inclua as funções de inicialização necessárias e a operação que retorne seu perímetro.

[http://scanuto.com/main\\_poligono.c](http://scanuto.com/main_poligono.c)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "Poligono.h"
4
5 int main(){
6     float perimetro;
7     Poligono* c1;
8     float lados[]={3.1,4.4,5.3,6.3};
9     //float *lados;
10    //lados=malloc(4*sizeof(float));
11    //lados[0]=3.1; lados[1]=4.4; lados[2]=5.3; lados[3]=6.3;
12    c1=poligono_cria(lados, 4);
13
14    perimetro=poligono_perimetro(c1);
15    printf("O perimetro do poligono eh: %f\n", perimetro);
16    poligono_libera(c1);
17    return 0;
18
19 }
```

# Criando uma TAD

Desenvolva um TAD que represente um polígono. Inclua as funções de inicialização necessárias e a operação que retorne seu perímetro.

<http://scanuto.com/Poligono.h>

```
1 //arquivo Poligono.h
2
3 typedef struct poligono Poligono;
4
5 //Cria um novo poligono
6 Poligono* poligono_cria(float *a, int qtdl);
7
8 //libera poligono
9 void poligono_libera(Poligono* c);
10
11 //calcula o perimetro de um poligono
12 float poligono_perimetro(Poligono* c);
```

# Vetor de pontos

Utilizando os conceitos de ponteiros e ponteiros de ponteiros, modifique o código (em <http://scanuto.com/vetorPontos.zip> ) implementando as funções `cria_vetor_pontos`, `imprime_media` e `libera_vetor_pontos` para tornar o código abaixo funcional:

```
1#include <stdio.h>
2#include <stdlib.h>
3#include "Ponto.h"
4int main(){
5    printf("criando vetor de 4 posicoes\n");
6
7    cria_vetor_pontos(4);
8
9    printf("inserindo 4 pontos\n");
10  pto_inserere(1,2);
11  pto_inserere(2,4);
12  pto_inserere(3,6);
13  pto_inserere(4,8);
14
15
16   //imprime o ponto médio
17   imprime_media();
18
19   libera_vetor_pontos();
20
21   return 0;
22 }
```

# Vetor de pontos

Utilizando os conceitos de ponteiros e ponteiros de ponteiros, modifique o código (em <http://scanuto.com/vetorPontos.zip> ) implementando as funções `cria_vetor_pontos`, `imprime_media` e `libera_vetor_pontos` para tornar o código abaixo funcional:

```
1#include <stdlib.h>
2#include <math.h>
3#include "Ponto.h" //inclui os Protótipos
4#include <stdlib.h>
5#include <stdio.h>
6
7//Definição do tipo de dados
8struct ponto{
9    float x;
10   float y;
11};
12
13//Cria um vetor de pontos do tipo *Ponto
14Ponto **vecPontos; //vetor de armazenamento dos ponteiros dos pontos
15int pontos_inseridos; //quantidade pontos inseridos ate o momento
16
17//inicializa o vetor de pontos, alocando a memoria
18void cria_vetor_pontos(int tamanho){
19    //preencher aqui com seu codigo...
20}
21
22void pto_inseri(float x, float y){
23    //preencher aqui com seu codigo...
24}
25
26//Libera toda a memoria alocada
27void libera_vetor_pontos(){
28    //preencher aqui com seu codigo...
29}
30
31//imprime o ponto medio da lista de pontos
32void imprime_media(){
33    //insira aqui o seu codigo...
34}
```

# Vetor

Utilizando  
implement

```
1 #include <stdlib.h>
2 #include <math.h>
3 #include "Ponto.h" //inclui os Protótipos
4 #include <stdlib.h>
5 #include <stdio.h>
6
7 //Definição do tipo de dados
8 struct ponto{
9     float x;
10    float y;
11};
12
13 //Cria um vetor de pontos do tipo *Ponto
14 Ponto **vecpontos; //vetor de armazenamento dos ponteiros dos pontos
15 int pontos_inseridos; //quantidade pontos inseridos ate o momento
16
17 //inicializa o vetor de pontos, alocando a memoria
18 void cria_vetor_pontos(int tamanho){
19 //preencher aqui com seu codigo...
20 }
21
22 void pto_insere(float x, float y){
23 //preencher aqui com seu codigo...
24 }
25
26 //Libera toda a memoria alocada
27 void libera_vetor_pontos(){
28 //preencher aqui com seu codigo...
29 }
30
31 //imprime o ponto medio da lista de pontos
32 void imprime_media(){
33 //insira aqui o seu codigo...
34 }
```

[n/vetorPontos.zip](#) )  
jo abaixo funcional:

# lista de pontos

- Na prática, há algum problema em utilizar a implementação de lista de pontos anterior se o usuário precisar de tratar múltiplas listas de pontos? Caso haja, o que precisaríamos fazer aprimorar tal implementação?
- Como implementar a função abaixo?
  - `void pto_remove(float posicao_ponto)`

# Referências

Estrutura de Dados descomplicada em Linguagem C (André Backes): Cap 5;

Vídeo aulas:

<https://programacaodescomplicada.wordpress.com/indice/estrutura-de-dados/>

Implementações:

<http://www.facom.ufu.br/~backes/wordpress/ProjPonto.zip>